Lezione 13 - Valutazione delle Prestazioni

- **Le prestazioni sono in unità di "oggetti-per-secondo"**
  - "grande" è meglio
- **Se siamo soprattutto interessati al tempo di risposta**
  - $P_X = \frac{1}{E_X}$
  - P_X = prestazioni della macchina X
  - E_X = tempo di esecuzione della macchina X
- **Speed up rispetto a una macchina di riferimento**
  - $S_{XY} = \frac{P_X}{P_Y} = \frac{E_Y}{E_X}$
  - Lo speed up di X rispetto a Y dice "X è S_{XY} volte più veloce di Y"
  - In percentuali, $s_{xy}=(S_{XY}-1)*100$ dice "X è s_{xy}% più veloce di Y"
- **Esempio:**
  - E_X = 1sec, E_Y = 1.2sec $\Rightarrow$ S_{XY} = 1.2 $\Rightarrow$ X è 1.2 volte più veloce di Y
    s_{xy} = 20% $\Rightarrow$ X è il 20% più veloce di Y

Vorrei un solo numero che mi indica le prestaz. della macchina

più piccolo è e meglio è

Prestaz. macchina X = 1 / Tempo di esecuzione del nostro

del programma

Dipende dal programma e meglio confrontarle

rapporto tempi esecuzione, se > 1 X è più veloce rispetto a Y

qual è la macchina di riferimento?

qual è la macchina di riferimento?

Qual è il tempo di esecuzione del programma? Cosa misura? Da cosa dipende?

Tempo Totale e Tempo di CPU

Uso come r. per misurare il tempo

- **T_J = Tempo Totale (o Tempo Assoluto o Tempo di Risposta o Tempo Trascorso (Elapsed Time))**
  - **tempo necessario per completare un lavoro; comprende una serie di tempi che sono "allungati" dalle interferenze di altri programmi che generano attività come:**
    - **accesso ai dischi**
    - **accesso alla memoria**
    - **attività di I/O**
    - **sovraccarico del Sistema Operativo**

Può includere tempi dovuti a ritardi esterni. Tempi che dipendono da cose esterne. } dipendono anche da altri ritardi esterni

- **T_{CPU} = Tempo di CPU (Tempo di Esecuzione del programma)**
  - **Tempo durante il quale un processore ha lavorato su un singolo programma**
    - **senza tempo speso in I/O (può includere il tempo "dell'uomo")**
    - **senza tempo per eseguire altri programmi**
  - **Può essere suddiviso in**
    - **T_U = tempo di CPU relativo all'utente**
    - **T_K = tempo di CPU relativo al Sistema Operativo (K=Kernel)**

Quantità più precisa. Filtrando i tempi aggiuntivi come I/O ecc. eliminando Tempi spesi per i/o ecc

del mio programma

es. sys. op.

macchina non può fare a meno del S.O.

macchina non può fare a meno del Kernel (sys op.)

$$T_U + T_K = T_{CPU}$$

istruzioni eseguite del programma nostro

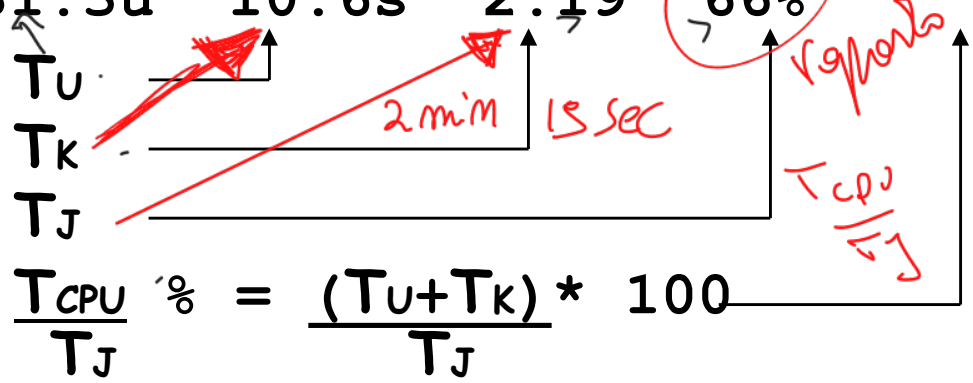
Tempo speso dal sistema operativo per eseguire richieste del nostro

Tempo di CPU misurato in UNIX/LINUX

- Dato il programma `myprogram`

```
time <myprogram>
```

```
81.3u   10.6s   2:19   66%
```



$$\frac{T_{CPU}}{T_J} \% = \frac{(T_U + T_K)}{T_J} * 100$$

Per il 34 % la macchina è usata
- per I/O
- per altri programmi
- per altra attività di sistema

- TK spesso non viene incluso
  - perché può nascondere inaccuratezze di misura
  - il calcolo può risultare sbilanciato se si usano sistemi operativi diversi
  - su alcune macchine il codice può essere considerato di kernel e su altre di utente
  - d'altra parte nessun programma può essere eseguito senza usare qualche parte di kernel

- Prestazioni di Sistema $\Rightarrow T_J$ Prestazioni CPU $\Rightarrow T_U$

Equazione delle prestazioni

$$T_{CPU} = C_{CPU} \cdot T_C = \frac{C_{CPU}}{fc} = N_{CPU} \cdot \overline{CPI} \cdot T_C$$

- T_C è il periodo di clock (fc è la frequenza di clock)
- C_{CPU} sono i cicli di clock
- N_{CPU} è il Numero di istruzioni eseguite **DINAMICAMENTE**
- $\overline{CPI}$ è il numero di Cicli Per Istruzione (medio)

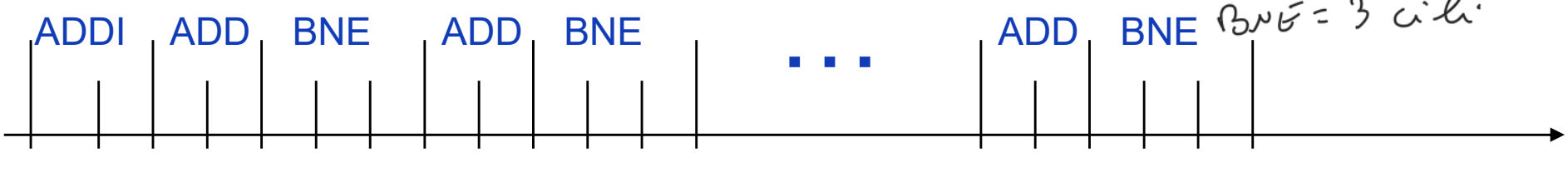
Esempio:

bne → 2 cicli
add → 1 ciclo
2 istruzioni, 3 cicli → /CPI=3/2=1.5

Esempio 1

Assumiamo t0=0, t1=0 :

```
        addi  t1, t1, 100
loop:   add   t0, t0, 1
        bne   t0, t1, loop
```



$C_{CPU}=502$, $N_{CPU}=201$

/ CPI= 502/201 ≈ 2.5

- **Per ottenere maggiori prestazioni si può:**
  - **ridurre i cicli di clock per un programma**
  - **aumentare la frequenza di clock**
- **Spesso però l'abbassamento del numero di cicli comporta invece un abbassamento della frequenza di clock**

Esempio 2

- $T_{CPU}(A) = 10s$ $f_C(A) = 400$ MHz ← MACCHINA A
- $T_{CPU}(B) = 6s$ $f_C(B) = ?$ ← MACCHINA B

Supponiamo che in B si riesca a ridurre il tempo ma con il risultato di avere

$$C_{CPU}(B) = 1.2 * C_{CPU}(A)$$

- Quanto deve essere la frequenza di clock della macchina B affinché il tempo di esecuzione del programma sia 6s?

- Soluzione:

$$C_{CPU}(A) = (f_C * T_{CPU})|_A = 4 * 10^9$$

$$f_C(B) = (C_{CPU} / T_{CPU})|_B = (1.2 * 4 * 10^9 / 6) = 800 \text{ MHz}$$

- Per ottenere una diminuzione del 40% del tempo di esecuzione è necessario raddoppiare la frequenza di clock

Esempio 3

- $f_c(A)$ = 1 GHz $f_c(B)$ = 500 MHz
- /CPI(A) = 2.5 /CPI(B) = 1.2
- Quale calcolatore è più veloce?
  (il set di istruzioni rimane lo stesso)

- $S_{AB} = \frac{T_{CPU}(B)}{T_{CPU}(A)} = \frac{N_{CPU}(B)*/CPI(B)*f_c(A)}{N_{CPU}(A)*/CPI(A)*f_c(B)} = \frac{1.2*1000}{2.5*500}$
- set istruzioni uguale → $N_{CPU}(A) = N_{CPU}(B)$
- S_{AB}= 0.9 → B è 1.1 volte più veloce di A

Motivi delle dipendenze dei parametri fondamentali

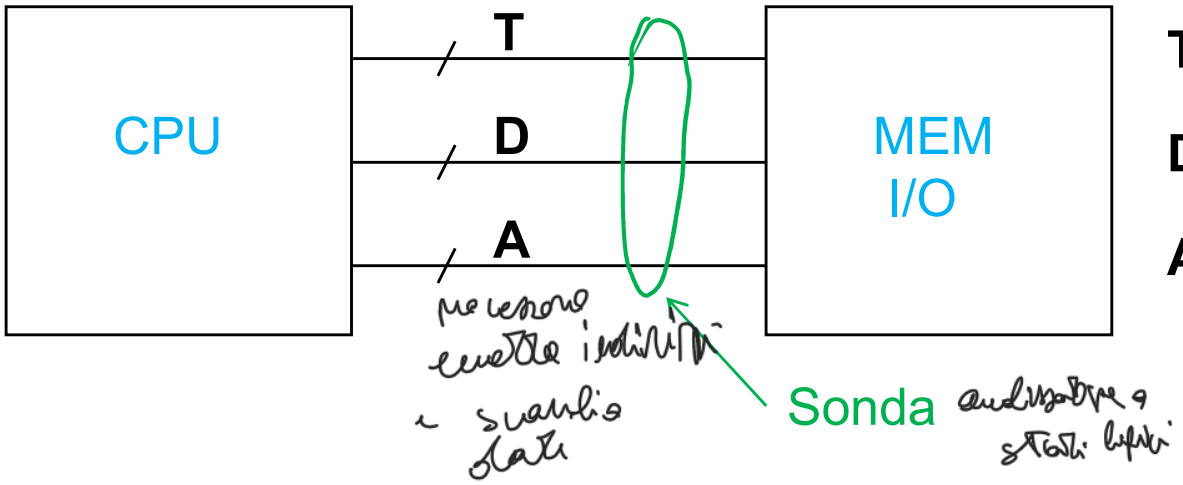
- **Programma**
  - **Fibonacci-ricorsivo vs. Fibonacci-iterativo → N_{CPU}**
- **Compilatore**
  - **Opzioni -O1, -O2, -O3 → N_{CPU}, /CPI**
- **Sistema Operativo**
  - **Una syscall può essere implementata diversamente in Linux rispetto a Windows → N_{CPU}**
- **Instruction Set (ISA)**
  - **RISC-V vs. x86 vs. ARM, #cicli necessari per es. add → N_{CPU}, /CPI**
- **Struttura CPU**
  - **Una pipeline elabora più istruzioni per ciclo → /CPI**
  - **Stadi (logica combinatoria) con meno porte logiche → maggiore fc**
- **Sottosistema di memoria**
  - **Es. add → 1 ciclo, lw → 100 cicli**
- **Tecnologia (elettronica)**
  - **Es. 14 nm invece che 90nm**

dipende da	N_{CPU}	/CPI	fc
Programma	X		
Compilatore	X	(X)	
Sistema Operativo	X		
ISA	X	X	
Struttura CPU		X	X
Sottosistema di Memoria		X	
Tecnologia			X

T_{CPU}	→ orologio
fc	→ dato di targa
N_{CPU}, /CPI	→ difficili da misurare

$$T_c = \frac{1}{fc}$$

Come valutare NCPU (1) – Sonda HW



T = TYPE={IF,DR,DW}

FETCH DATA-READ DATA-WRITE

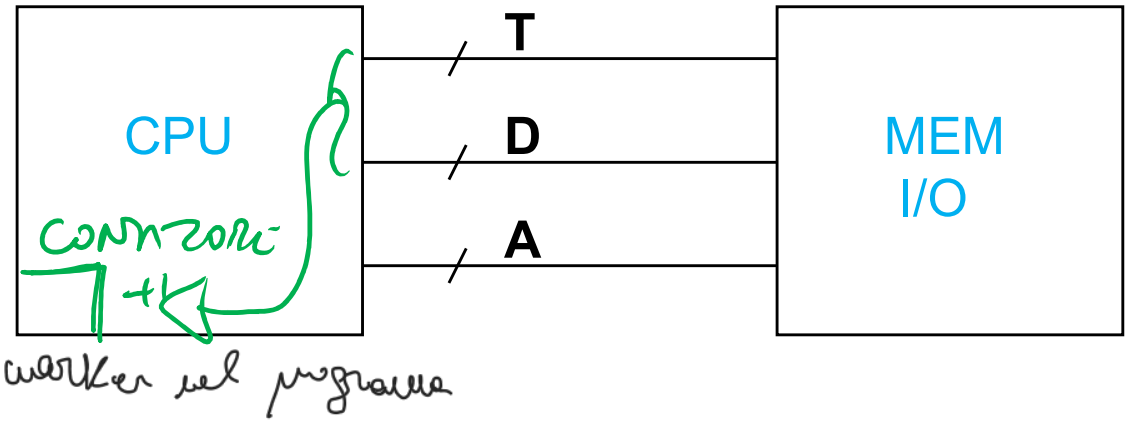
D = DATA

A = ADDRESS

- **Strumentazione Hardware (sonda o probe) che legge le istruzioni che il processore preleva dalla memoria**

- **Richiede di poter aprire la macchina**
- **Richiede la disponibilità di una opportuna sonda**
- **Richiede la disponibilità di un analizzatore di stati logici**

Come valutare NCPU (2) – Contatori HW



T = TYPE={IF,DR,DW}
FETCH DATA-READ DATA-WRITE

D = DATA

A = ADDRESS

- Contatori Hardware che contino il numero di istruzioni
  - Disponibili sui processori Intel (ed Alpha)

- Supporto da parte del processore e del sistema
  - Alpha: ATOM tools, Intel VTUNE

Come valutare NCPU (3) – Instrumentazione SW (Sonda-SW)

Codice Originale

```
add t0,t1,t2      #1ciclo
lw  t0,0(t1)      #100cicli
bne t3,t4,Loop    #3cicli
```

Codice Instrumentation

```
add  s8,zero,3
add  s9,zero,104
add  t0,t1,t2
lw   t0,0(t1)
bne  t3,t4,Loop
```

Inserisco conteggio istruz. e conteggio di N cicli

1) Riservo un registro (es. s8) per contare le istruzioni
2) Riservo un registro (es. s9) per contare i cicli di ogni basic-block*
3) Inserisco due istruzioni per tali conteggi per ogni basic-block* del programma (instrumentazioni)
4) Al termine del programma, s8 contiene Ncpu mentre s9/s8 è pari a /CPI

cicli / istruz.

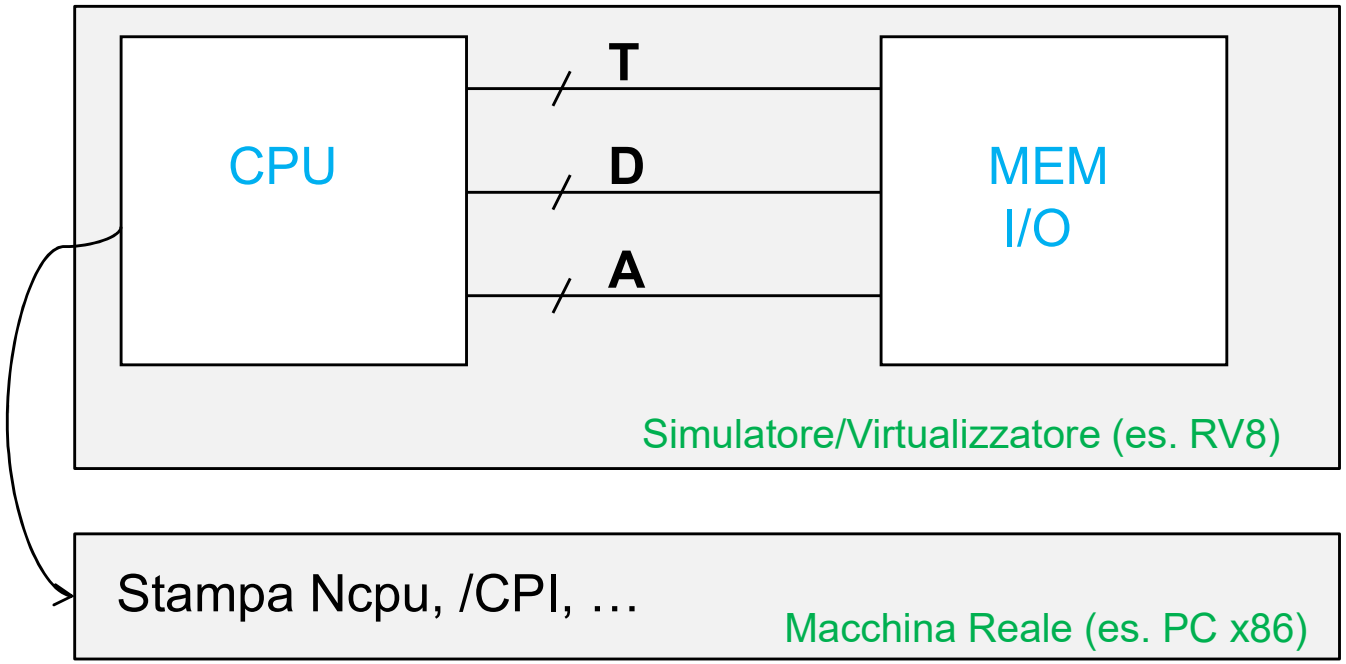
- Metodo usato in vari processori
  - MIPS R4400: Pixie Alpha: Atom Intel: VTUNE

- Istrum. dice quale è il Basic block più presente ecc., quale da ottimizzare ecc.

Cicli necessari per programma se non ci fossero le 2 add/

*Basic-block=gruppo di istruzioni con un solo punto di ingresso e un solo punto di uscita

Come valutare NCPU (4) – Virtual-Machine SW



- **Usare dei simulatori/virtualizzatori/emulatori**
  - **RV8 → simulatore di processore RISC-V**
  - **SPIM → simulatore di processore MIPS**
  - **QEMU → emulatore molto diffuso sotto Linux → virtual machine**
  - **Virtualbox, Vmware, VirtualPC → ulteriori virtual machines**

Come calcolare /CPI

$$\overline{CPI} = \sum_{k=1}^{\text{Tipi}} \frac{N_{CPU,k} \cdot CPI_k}{N_{CPU}}$$

Clock medio per istr.

Si può calcolare da quante istruzioni di ogni tipo vengono eseguite ($N_{CPU,K}$)

$$CICLI = N_{CPU} \cdot \overline{CPI}$$

$$CICLI = \sum_{K=1}^{T} N_{CPU,K} \cdot CPI_K$$

$$\overline{CPI} = \sum_{k=1}^{T} \overline{CPI}_k = \sum_{k=1}^{T} p_{CPU,k} \cdot CPI_k = \sum_{k=1}^{T} \frac{N_{CPU,k}}{N_{CPU}} \cdot CPI_k$$

- T = numero tipi diversi di istruzioni
- $N_{CPU,K}$ = numero di istruzioni di tipo K
- $p_{CPU,k}$ = frequenza relativa di quel tipo di istruzione $\equiv N_{CPU,K}/N_{CPU}$
- CPI_K = CPI delle istruzioni di tipo K
- $/CPI_K$ = CPI **MEDIO** delle istruzioni di tipo K $\equiv p_{CPU,k} * CPI_K$

Op	$p_{CPU,K} \equiv N_{CPU,K}/N_{CPU}$	CPI_K	$/CPI_K \equiv p_{CPU,k} * CPI_K$	$\%T_{CPU,k} = CICLI_k / CICLI$ $/CPI_K : /CPI$
Arithmetic	0.50	1	0.5	33%
Load	0.20	2	0.4	27%
Store	0.10	2	0.2	13%
Branch	0.20	2	0.4	27%

/CPI = 0.5+0.4+0.2+0.4 = 1.5

Scalabilità

Capacità di un sistema a seguire un miglioramento prestaz. scalabili rispetto ad un parametro, es. freq con la freq

$$P := \textit{Prestazioni} \qquad P \propto 1/T_{CPU} = \frac{1}{N_{CPU} \cdot \overline{CPI}} \cdot f_C$$

prestaz. inversa Tempo per Teo. se raddoppio freq. raddoppio prestaz.

- Diciamo che un sistema ha **prestazioni scalabili con la frequenza** se, con $f_{200}/f_{100}=2$, ottengo $P_{200}/P_{100}=2$ es. raddoppiando fc, raddoppio prestaz.
  - Abbiamo visto però che, aumentando la frequenza, è verosimile che il CPI aumenti... es. $CPI_{200}=1.2*CPI_{100}$; in tal caso si otterrebbe $P_{200}/P_{100}=f_{200}/f_{100}*CPI_{100}/CPI_{200}=2/1.2\approx1.6$ quindi *poca scalabilità* (<2)

raddoppio clock, non do doppio velocità

- Questo è ciò che si è verificato storicamente nei Pentium
  - Per es., se il sottosistema di memoria 'non tiene il passo del processore' architettura superscalare
  - Risolto nei PentiumPro mantenendo uno stesso CPI all'aumentare di f_C
- La capacità di fornire maggiori prestazioni al variare di un parametro viene chiamata *scalabilità rispetto a quel parametro*
  - Per es., il processore PentiumPro risulta *più scalabile* del processore Pentium *con la frequenza* ($Scalabilita`(f)_{PentiumPro} > Scalabilita`(f)_{Pentium}$)
- Diciamo che un sistema ha *scalabilità ideale lineare* se $Scalabilita`(p) = k \cdot p$ rispetto a p

se scalabilità avviene linearmente rispetto a quel parametro

dove (k=costante, p=parametro)

scalabilità rispetto ad un parametro, freq.

Legge di Amdhal

- **In generale può essere difficile stabilire l'esatta dipendenza da un dato parametro (anche quando questo compare esplicitamente nell'equazione!)**
- **Per questo motivo viene utilizzata la seguente formula, per mettere in relazione lo speedup di una parte del sistema con lo speedup dell'intero sistema:**

$$S = \frac{P_{dopo}}{P_{prima}} = \frac{T_{prima}}{T_{dopo}} = \frac{T_{prima}}{\frac{T_{migliorabile}}{a} + T_{NON-migliorabile}} = \frac{1}{\frac{p}{a} + (1-p)}$$

speedup prestazioni tempo

dove
p := parte del tempo 'migliorabile'=Tmigliorabile/Tprima
a := speed-up della parte di sistema migliorabile

Quindi: $0 \leq p \leq 1$

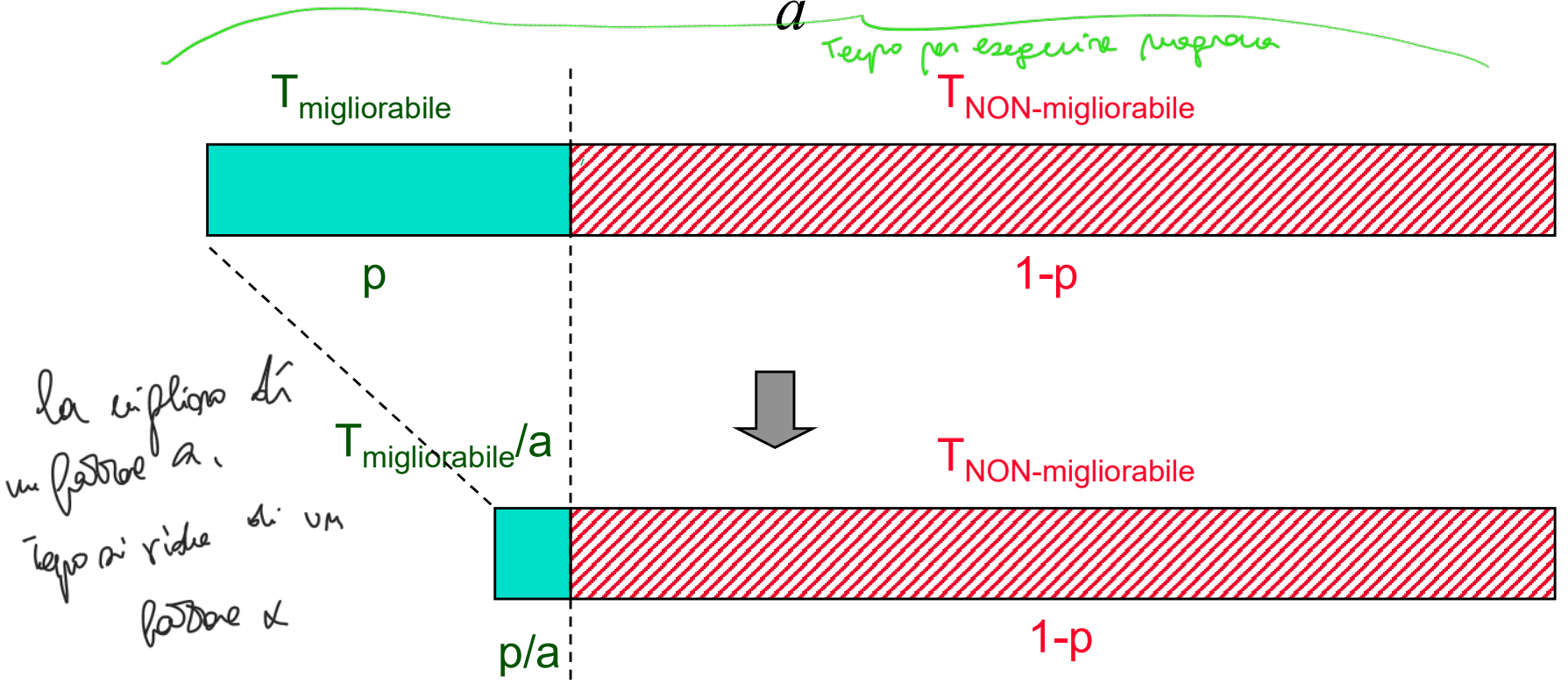
Mentre: $a \geq 1$,

Roberto Giorgi, Università di Siena, C124L13, Slide 15

Legge di Amdhal (2)

Dimostrazione "grafica" della legge di Amdhal

$$S = \frac{1}{\frac{p}{a} + (1-p)}$$

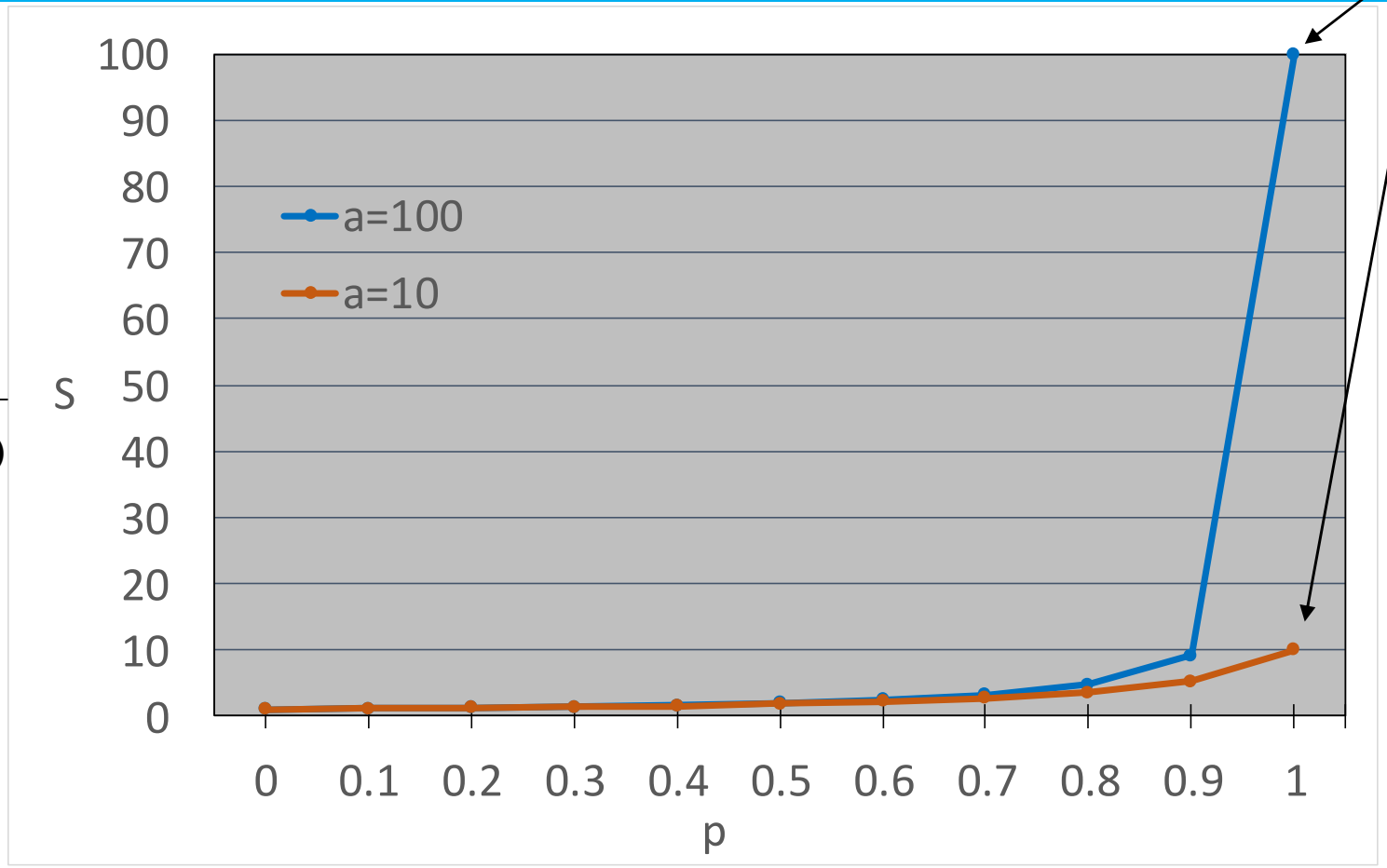


p=parte del tempo 'migliorabile'=$T_{migliorabile}/T_{prima}$

a=speed-up della parte di sistema migliorabile

Legge di Amdhal (3) – S al variare di p

$$S = \frac{1}{\frac{p}{a} + (1-p)}$$



Nota: questo punto (per p=1) vale 'a'

→ lo speed-up di prestazioni S assume valori significativamente alti solo quando la parte migliorata p è abbastanza grande (es. p>0.8)

→ è importante concentrare i miglioramenti su una parte del sistema che sia <u>consistente</u>

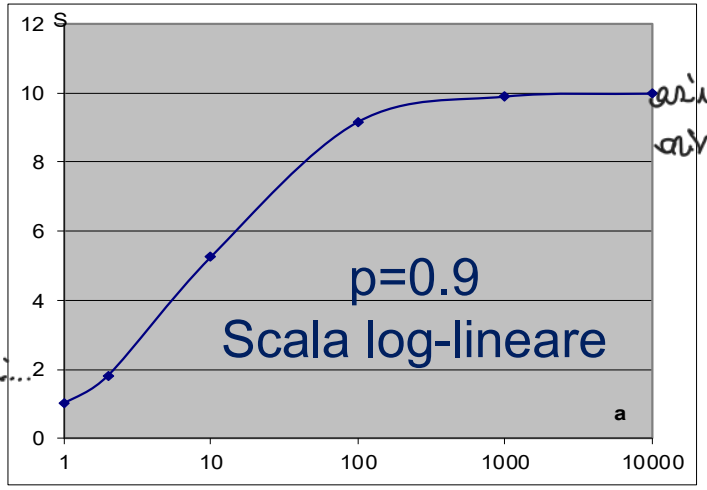
Legge di Amdhal (4) – S al variare di a

- **Supponiamo che il 90% (→p=0.9) di un programma possa essere eseguita in parallelo (es. aggiungendo più processori)**

$$S = \frac{1}{\frac{p}{a} + (1-p)}$$

$$\lim_{a \to \infty} S = \frac{1}{1-p}$$

a	S	p
1	1.000	0.9
2	1.818	0.9
10	5.263	0.9
100	9.174	0.9
1000	9.911	0.9
10000	9.991	0.9

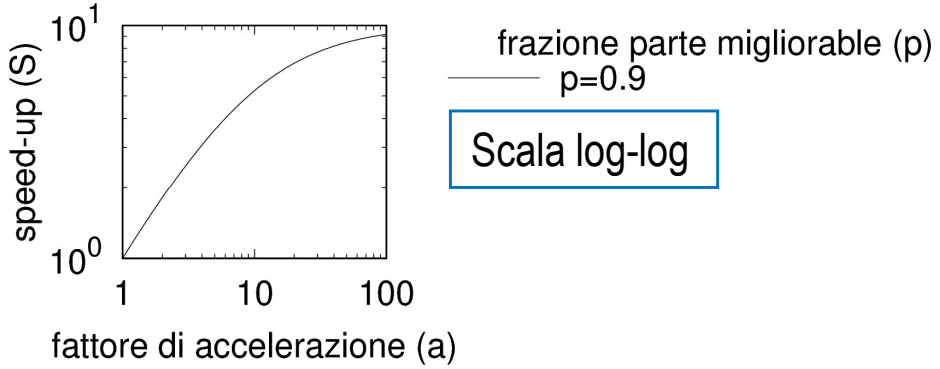


→lo speed-up di prestazioni S ottenibile migliorando una parte p è limitato da 1/(1-p)

Nell'esempio in figura, per a→ ∞ il massimo speed-up ottenibile è pari a 1/0.1=10. Quindi invece di aggiungere 9999 processori conviene parallelizzare anche una parte del restante 10%

Nota:

Se faccio lo stesso disegno in scala log-log:



Conclusioni dalla legge di Amdhal

1) è molto importante migliorare la parte più utilizzata del sistema!

e deve essere consistente

2) A un certo punto è importante anche migliorare la parte meno utilizzata del sistema…
(ovvero spingere il più possibile p a valori molto alti)

2) dobbiamo accelerarci anche di resto